Writing A Unix Device Driver

The book Operating System by Rohit Khurana is an insightful work that elaborates on fundamentals as well as advanced topics of the discipline. It offers an in-depth coverage of concepts, design and functions of an operating system irrespective of the hardware used. With illustrations and examples the aim is to make the subject crystal clear and the book extremely student-friendly. The book caters to undergraduate students of most Indian universities, who would find subject matter highly informative and enriching. Tailored as a guide for self-paced learning, it equips budding system programmers with the right knowledge and expertise. The book has been revised to keep pace with the latest technology and constantly revising syllabuses. Thus, this edition has become more comprehensive with the inclusion of several new topics. In addition, certain sections of the book have been thoroughly revised. Key Features • Case studies of Unix, Linux and Windows to put theory concepts into practice • A crisp summary for recapitulation with each chapter • A glossary of technical terms • Insightful questions and model test papers to prepare for the examinations New in this Edition • More types of operating system, like PC and mobile; Methods used for communication in client-server systems. • New topics like: Thread library; Thread scheduling; Principles of concurrency, Precedence graph, Concurrency conditions and Sleeping barber problem; Structure of page tables, Demand segmentation and Cache memory organization; STREAMS; Disk attachment, Stable and tertiary storage, Record blocking and File sharing; Goals and principles of protection, Access control matrix, Revocation of access rights, Cryptography, Trusted systems, and Firewalls.

A practical, hands-on guide to driver design and development. Writing UNIX Device Drivers in C contains all the information you need to design and build UNIX device drivers. Adams and Tondo introduce the concept that device drivers are the implementation of an abstract software architecture and present a template-based development process that reduces the drudgery of implementing and debugging. This approach shortens development time and allows you to focus on the problem the device driver is designed to solve.

Comprehensive and detailed, this book covering the UNIX System V is organized by application, allowing the reader to look up any particular topic and quickly learn how it is accomplished in a UNIX environment.

Nwely updated to include new calls and techniques introduced in Versions 2.2 and 2.4 of the Linux kernel, a definitive resource for those who want to support computer peripherals under the Linux operating system explains how to write a driver for a broad spectrum of devices, including character devices, network interfaces, and block devices. Original. (Intermediate)

Networking Device Drivers Sanjay Dhawan Although device drivers are an integral component of any network, little practical information has been available on their implementation. Networking Device Drivers fills this void by providing a wealth of in-depth information on a variety of networking device driver architectures, for a number of operating systems. The book begins with a discussion of the basics of networking, various operating systems — such as Windows 3.x, Windows NT, UNIX, DOS, OS/2, and NetWare — and their network architectures. It then provides up-to-date, specific information on a variety of device driver architectures including: Network Driver Interface Standard (NDIS2 and NDIS3) Open Data-Link Interface (ODI) Packet Driver Data Link Provider Interface (DLPI) The definitive "how-to" book, Networking Device Drivers provides ample detail in an easy-to-understand format, allowing readers to use the information from specific chapters to immediately begin work on device driver implementation. Whatever your level of network involvement, this book has something for you: MIS managers will find the extensive information on network architecture and implementation very useful for configuring and maintaining their own networks.

Network engineers will find the information about specific network drivers invaluable when designing network related hardware and software applications. Software engineers will find the entire book essential for writing device drivers for network interface cards. Network card manufacturers will find the book useful for teaching engineering staffs about networking device driver architectures. Computer science students will find the book offers a comprehensive introduction to the practical aspects of computer networks. No matter what your involvement with networks, Networking Device Drivers should be a part of your reference collection.

"The Encyclopedia of Microcomputers serves as the ideal companion reference to the popular Encyclopedia of Computer Science and Technology. Now in its 10th year of publication, this timely reference work details the broad spectrum of microcomputer technology, including microcomputer history; explains and illustrates the use of microcomputers throughout academe, business, government, and society in general; and assesses the future impact of this rapidly changing technology."

???????IEEE?????????—POSIX??(????Pthreads??),????????????????????????,????????????????????????????

Linux® is being adopted by an increasing number of embedded systems developers, who have been won over by its sophisticated scheduling and networking, its cost-free license, its open development model, and the support offered by rich and powerful programming tools. While there is a great deal of hype surrounding the use of Linux in embedded systems, there is not a lot of practical information. Building Embedded Linux Systems is the first in-depth, hard-core guide to putting together an embedded system based on the Linux kernel. This indispensable book features arcane and previously undocumented procedures for: Building your own GNU development toolchain Using an efficient embedded development framework Selecting, configuring, building, and installing a target-specific kernel Creating a complete target root filesystem Setting up, manipulating, and using solid-state storage devices Installing and configuring a bootloader for the target Cross-compiling a slew of utilities and packages Debugging your embedded system using a plethora of tools and techniques Details are provided for various target architectures and hardware configurations, including a thorough review of Linux's support for embedded hardware. All explanations rely on the use of open source and free software packages. By presenting how to build the operating system components from pristine sources and how to find more documentation or help, this book greatly simplifies the task of keeping complete control over one's embedded operating system, whether it be for technical or sound financial reasons.Author Karim Yaghmour, a well-known designer and speaker who is responsible for the Linux Trace Toolkit, starts by discussing the strengths and weaknesses of Linux as an embedded operating system. Licensing issues are included, followed by a discussion of the basics of building embedded Linux systems. The configuration, setup, and use of over forty different open source and free software packages commonly used in embedded Linux systems are also covered. uClibc, BusyBox, U-Boot, OpenSSH, thttpd, tftp, strace, and gdb are among the packages discussed.

New requirements for UNIX device drivers arise every week. These requirements range from drivers for mice to graphical

display cards, from point of sales terminals to intelligent telephone exchanges. Writing Device Drivers for SCO UNIX is based on a training course run by The Santa Cruz Operation Ltd. It is a practical guide that will equip you with the skills you need to meet the challenge of writing a variety of device drivers. You will explore: The structure and mechanisms of an operating system, the concept of device independence and computer peripheral architecture Numerous hands-on exercises. By working through these exercises you will . . . Write a device driver for a mouse Write a Stream driver Write a simple line discipline Experiment with interrupts Examples based on the best selling, most up to date version 3.2 V4 of SCO UNIX Principles that will enable you to extend your skills to writing device drivers for other operating systems. If you are a student or a professional systems programmer with some experience of using C and developing UNIX programs you will find this book an invaluable guide.

Offers practical, hands-on guidance in developing your own device drives. Clearly demonstrates how to write device drivers for adding disk drives, printers, magnetic tapes and other peripherals to your Unix system. Presents procedures for developing and testing new device drivers including how to select a convenient working directory; use make-files; preserve and boot alternative kernal versions; debug driver code and much more. Packed with examples which illustrate each operation in practice.

Mac OS X was released in March 2001, but many components, such as Mach and BSD, are considerably older. Understanding the design, implementation, and workings of Mac OS X requires examination of several technologies that differ in their age, origins, philosophies, and roles. Mac OS X Internals: A Systems Approach is the first book that dissects the internals of the system, presenting a detailed picture that grows incrementally as you read. For example, you will learn the roles of the firmware, the bootloader, the Mach and BSD kernel components (including the process, virtual memory, IPC, and file system layers), the object-oriented I/O Kit driver framework, user libraries, and other core pieces of software. You will learn how these pieces connect and work internally, where they originated, and how they evolved. The book also covers several key areas of the Intel-based Macintosh computers. A solid understanding of system internals is immensely useful in design, development, and debugging for programmers of various skill levels. System programmers can use the book as a reference and to construct a better picture of how the core system works. Application programmers can gain a deeper understanding of how their applications interact with the system. System administrators and power users can use the book to harness the power of the rich environment offered by Mac OS X. Finally, members of the Windows, Linux, BSD, and other Unix communities will find the book valuable in comparing and contrasting Mac OS X with their respective systems. Mac OS X Internals focuses on the technical aspects of OS X and is so full of extremely useful information and programming examples that it will definitely become a mandatory tool for every Mac OS

X programmer.

What is this book about? If you have some programming experience and are ready to venture into Linux programming, this updated edition of the bestselling entry-level book takes you there. The authors guide you step by step, using construction of a CD database application to give you hands-on experience as you progress from the basic to the complex. You'll start with fundamental concepts like writing Unix programs in C. You'll learn basic system calls, file I/O, interprocess communication, and shell programming. You'll become skilled with the toolkits and libraries for working with user interfaces. The book starts from the basics, explaining how to compile and run your first program. New to this edition are chapters on MySQL® access and administration; programming GNOME and KDE; and Linux standards for portable applications. Coverage of kernel programming, device drivers, CVS, grep, and GUI development environments has expanded. This book gives you practical knowledge for real wor ld application. What does this book cover? In this book, you will learn how to Develop programs to access files and the Linux environment Use the GNU compiler, debugger and other development tools Program data storage aapplications for MySQL and DBM database systems Write programs that take advantage of signals, processes and threads Build graphical user interfaces using both the GTK (for GNOME) and Qt (for KDE) libraries Write device drivers that can be loaded into the Linux kernel Access the network using TCP/IP sockets Write scripts that use grep, regular expressions and other Linux facilities Who is this book for? This book is for programmers with some C or C++ experience, who want to take advantage of the Linux development environment. You should have enough Linux familiarity to have installed and configured users on Linux.

Provides information on writing a driver in Linux, covering such topics as character devices, network interfaces, driver debugging, concurrency, and interrupts.

A device driver is used in the UNIX system to control specific peripheral devices, such as floppy disks or cartridge tapes. This is the first book to deal exclusively with writing device driver software, allowing UNIX users to expand their system's flexibility by creating their own device drivers for those not supported by the company marketing the system. In clear and concise language, it provides detailed examples of driver logic, development methods, special requirements, and steps to connecting device driver programs to a variety of systems. Includes numerous sample programs, and an appendix with program listings for all examples.

Writing UNIX Device DriversAddison-Wesley Professional

Pajari provides application programmers with definitive information on writing device drivers for the UNIX operating system. The comprehensive coverage includes the four major categories of UNIX device drivers: character, block, terminal, and stream drivers. (Operating Systems)

For users of the Digital UNIX (formerly DEC OSF/1) operating system, as well as for systems engineers interested in writing UNIX-based device drivers. Discusses how to write device drivers for computer systems running the Digital UNIX operating system. In addition, the volume provides information on designing drivers, UNIX-based data structures, and OSF-based kernel interfaces. Annotation copyright by Book News, Inc., Portland, OR

New for UNIX System V Release 4.2, this guide contains the latest information for writing, installing and testing UNIX System V device drivers. It provides an in-depth explanation of new SVR4.2 features such as dynamically loadable kernel modules, the new device driver installation tools and the new system configuration file formats.

For more than 40 years, Computerworld has been the leading source of technology news and information for IT influencers worldwide. Computerworld's award-winning Web site (Computerworld.com), twice-monthly publication, focused conference series and custom research form the hub of the world's largest global IT media network.

For developers who must know and understand the fundamentals to be able to apply the more advanced aspects that will emerge with NT 5, here is an in-depth book to the rescue, covering the core techniques of programming NT device drivers.

A growing concern of mine has been the unrealistic expectations for new computer-related technologies introduced into all kinds of organizations. Unrealistic expectations lead to disappointment, and a schizophrenic approach to the introduction of new technologies. The UNIX and real-time UNIX operating system technologies are major examples of emerging technologies with great potential benefits but unrealistic expectations. Users want to use UNIX as a common operating system throughout large segments of their organizations. A common operating system would decrease software costs by helping to provide portability and interoperability between computer systems in today's multivendor environments. Users would be able to more easily purchase new equipment and technologies and cost-effectively reuse their applications. And they could more easily connect heterogeneous equipment in different departments without having to constantly write and rewrite interfaces. On the other hand, many users in various organizations do not understand the ramifications of general-purpose versus real-time UNIX. Users tend to think of "real-time" as a way to handle exotic heart-monitoring or robotics systems. Then these users use UNIX for transaction processing and office applications and complain about its performance, robustness, and reliability. Unfortunately, the users don't realize that real-time capabilities added to UNIX can provide better performance, robustness and reliability for these non-real-time applications. Many other vendors and users do realize this, however. There are indications even now that general-purpose UNIX will go away as a separate entity. It will be replaced by a real-time UNIX. General-purpose UNIX will exist only as a subset of real-time UNIX.

Discover how to write high-quality character driver code, interface with userspace, work with chip memory, and gain an in-depth understanding of working with hardware interrupts and kernel synchronization Key Features: Delve into hardware interrupt handling, threaded IRQs, tasklets, softirqs, and understand which to use when Explore powerful techniques to perform user-kernel interfacing, peripheral I/O and use kernel mechanisms Work with key kernel synchronization primitives to solve kernel concurrency issues Book Description: Linux Kernel Programming Part 2 - Char Device Drivers and Kernel Synchronization is an ideal companion guide to the Linux Kernel Programming book. This book provides a comprehensive introduction for those new to Linux device driver development and will have you up and running with writing misc class character device driver code (on the 5.4 LTS Linux kernel) in next to no time. You'll begin by learning how to write a simple and complete misc class character driver before interfacing your driver with user-mode processes via procfs, sysfs, debugfs, netlink sockets, and ioctl. You'll then find out how to work with hardware I/O memory. The book covers working with hardware interrupts in depth and helps you understand interrupt request (IRQ) allocation, threaded IRQ handlers, tasklets, and softirqs. You'll also explore the practical usage of useful kernel mechanisms, setting up delays, timers, kernel threads, and workqueues. Finally, you'll discover how to deal with the complexity of kernel synchronization with locking technologies (mutexes, spinlocks, and atomic/refcount operators), including more advanced topics such as cache effects, a primer on lock-free techniques, deadlock avoidance (with lockdep), and kernel lock debugging techniques. By the end of this Linux kernel book, you'll have learned the fundamentals of writing Linux character device driver code for real-world projects and products. What You Will Learn: Get to grips with the basics of the modern Linux Device Model (LDM) Write a simple yet complete misc class character device driver Perform user-kernel interfacing using popular methods Understand and handle hardware interrupts confidently Perform I/O on peripheral hardware chip memory Explore kernel APIs to work with delays, timers, kthreads, and workqueues Understand kernel concurrency issues Work with key kernel synchronization primitives and discover how to detect and avoid deadlock Who this book is for: An understanding of the topics covered in the Linux Kernel Programming book is highly recommended to make the most of this book. This book is for Linux programmers beginning to find their way with device driver development. Linux device driver developers looking to overcome frequent and common kernel/driver development issues, as well as perform common driver tasks such as user-kernel interfaces, performing peripheral I/O, handling hardware interrupts, and dealing with concurrency will benefit from this book. A basic understanding of Linux kernel internals (and common APIs), kernel module development, and C programming is required.

?????????Linux????????????,???????????????????,???????Intel?????????

Device drivers literally drive everything you're interested in--disks, monitors, keyboards, modems--everything outside the computer chip and memory. And writing device drivers is one of the few areas of programming for the Linux operating system that calls for unique, Linux-specific knowledge. For years now, programmers have relied on the classic Linux Device Drivers from O'Reilly to master this critical subject. Now in its third edition, this bestselling guide provides all the information you'll need to write drivers for a wide range of devices.Over the years the book has helped countless programmers learn: how to support computer peripherals

under the Linux operating system how to develop and write software for new hardware under Linux the basics of Linux operation even if they are not expecting to write a driver The new edition of Linux Device Drivers is better than ever. The book covers all the significant changes to Version 2.6 of the Linux kernel, which simplifies many activities, and contains subtle new features that can make a driver both more efficient and more flexible. Readers will find new chapters on important types of drivers not covered previously, such as consoles, USB drivers, and more.Best of all, you don't have to be a kernel hacker to understand and enjoy this book. All you need is an understanding of the C programming language and some background in Unix system calls. And for maximum ease-of-use, the book uses full-featured examples that you can compile and run without special hardware.Today Linux holds fast as the most rapidly growing segment of the computer market and continues to win over enthusiastic adherents in many application areas. With this increasing support, Linux is now absolutely mainstream, and viewed as a solid platform for embedded systems. If you're writing device drivers, you'll want this book. In fact, you'll wonder how drivers are ever written without it.

A full explanation of the STREAMS I/O facilities, this guide details how to use those facilities for writing UNIX System V kernel modules and device drivers. STREAMS is a general, flexible facility for the development of input/output services in UNIX System V. This book is a comprehensive guide to STREAMS for network and system programmers, including the latest information on: STREAMS programming interfaces; STREAMS in a multiprocessing environment; STREAMS drivers and multiplexors; STREAMS debugging and utilities.

bull; Learn UNIX essentials with a concentration on communication, concurrency, and multithreading techniques bull; Full of ideas on how to design and implement good software along with unique projects throughout bull; Excellent companion to Stevens' Advanced UNIX System Programming

The definitive source of information for kernel-level STREAMS programming--in both uniprocessor and multiprocessor UNIX System V Release 4 environments. This guide is an indispensable resource for network and systems programmers responsible for designing and writing STREAMS-based modules and device drivers.

??????????

This comprehensive reference consists of two parts. The first part describes the Device Driver Interface/Driver-Kernel Interface (DDI/DKI). The second part describes routines of the Portable Device Interface (PDI). Intended for programmers, software developers and administrators working with UNIX System V Release 4 or later.

The only book available on networking device drivers, this book describes the various network device driver architectures and covers the most common ones in great detail--including NDIS, 3COM and Microsoft; ODI from Novell; Packet Driver from Ftp Software; and DLPI from USL, Inc. Popular network operating systems are also covered from the device driver standpoint.

Having already helped two generations of programmers explore Linux and write devices, the fourth edition of this classic book delves into tty, USB, and HCI devices such as keyboards, in addition to basic character devices. Linux Device Drivers includes numerous full-featured examples that you can compile and run without special hardware. Written by well-known leaders in Linux development and programming, this book covers significant changes to Version 3.2 of the Linux kernel, the basis of the Precise Pangolin release of Ubuntu. All you need to get

started is an understanding of the C programming language and some background in Unix system calls. Learn how to support computer peripherals under the Linux operating system Develop and write software for new hardware that Linux supports Understand the basics of Linux operation, even if you don't expect to write a driver Dive into new chapters on video, audio, wireless, and Bluetooth devices As the operating system for Android and many embedded systems, Linux constantly needs new device drivers. This book helps you get it done.
An authoritative guide to Windows NT driver development, now completely revised and updated. The CD-ROM includes all source code, plus Microsoft hardware standards documents, demo software, and more.